**Encapsulation**

In an object oriented python program, you can restrict access to methods and variables. This can prevent the data from being modified by accident and is known as encapsulation.

We can not call the private methods from the outside of the class using object.

Encapsulation is only for preventing the access of methods and variables from accidentally but not intentionally.

We can make any methods and variables private by double underscore (\_\_).

**Ex:**

**class** Car:

**def** \_\_init\_\_(self):

self.\_\_updateSoftware()

**def** drive(self):

**print** 'driving'

**def** \_\_updateSoftware(self):

**print** 'updating software'

redcar = Car()

redcar.drive()

*#redcar.\_\_updateSoftware() not accesible from object.*

Variables can be private which can be useful on many occasions. A private variable can only be changed within a class method and not outside of the class.

Objects can hold crucial data for your application and you do not want that data to be changeable from anywhere in the code.

**class** Car:

\_\_maxspeed = 0

\_\_name = ""

**def** \_\_init\_\_(self):

self.\_\_maxspeed = 200

self.\_\_name = "Supercar"

**def** drive(self):

**print** 'driving. maxspeed ' + str(self.\_\_maxspeed)

redcar = Car()

redcar.drive()

redcar.\_\_maxspeed = 10 *# will not change variable because its private*

redcar.drive()

|  |
| --- |
|  |

If you want to change the value of a private variable, a setter method is used.  This is simply a method that sets the value of a private variable.

|  |
| --- |
| **class** Car:    \_\_maxspeed = 0  \_\_name = ""    **def** \_\_init\_\_(self):  self.\_\_maxspeed = 200  self.\_\_name = "Supercar"    **def** drive(self):  **print** 'driving. maxspeed ' + str(self.\_\_maxspeed)    **def** setMaxSpeed(self,speed):  self.\_\_maxspeed = speed    redcar = Car()  redcar.drive()  redcar.setMaxSpeed(320)  redcar.drive() |

| **Type** | **Description** |
| --- | --- |
| public methods | Accessible from anywhere |
| private methods | Accessible only in their own class. starts with two underscores |
| public variables | Accessible from anywhere |
| private variables | Accesible only in their own class or by a method if defined. starts with two underscores |

Other programming languages have protected class methods too, but Python does not.

Encapsulation gives you more control over the degree of coupling in your code, it allows a class to change its implementation without affecting other parts of the code.

**Polymorphism:**

We have two types of polymorphism(Many Forms)

1.Method Overloading

2.Method Overriding

#Method overloading is not possible in the python.

#Method overriding is possible

**Example:**

class emp\_details():

def cook(self):

print ("This is cook method")

class one(emp\_details):

def cook(self):

print ("This is cool method in the method one")

class second(emp\_details):

def cook(self):

print ("This is cool method in the seconde method")

obj=one()

obj1=second()

obj.cook()

obj1.cook()

**Output:**

This is cool method in the method one

This is cool method in the second method

**Inheritance:**

It means child class inherits the properties of parent class. Mainly inheritance is used for code reusability.

class parent():

class child(parent):

Here child class inherits the properties of parent class.

**Abstraction:**

Use the “[abc](http://docs.python.org/library/abc.html)” module to create abstract classes. Use the “[abstractmethod”](https://docs.python.org/library/abc.html#abc.abstractmethod) decorator to declare a method abstract.

import abc

from abc import ABC, abstractmethod

class AbstractClass(ABC):

def \_\_init\_\_(self, value):

self.value = value

super().\_\_init\_\_()

@abstractmethod

def eat(self):

pass

class Parents(AbstractClass):

def eat(self):

return "eat solid food "+ str(self.value) + " times each day"

class Babies(AbstractClass):

def eat(self):

return "Milk only "+ str(self.value) + " times or more each day"

food = 3

mom = Parents(food)

print("moms ----------")

print(mom.eat())

infant = Babies(food)

print("infants ----------")

print(infant.eat())

**OUTPUT:**

moms ----------

eat solid food 3 times each day

infants ----------

Milk only3 times or more each day